Introducción a los patrones

1. Que son los patrones?

Los patrones de software son soluciones reutilizables a los problemas que ocurren durante el desarrollo de un sistema de software o aplicación. Estos proporcionan un proceso consistente o diseño que uno o más desarrolladores pueden utilizar para alcanzar sus objetivos. También proporciona una arquitectura uniforme que permite una fácil expansión, mantenimiento y modificación de una aplicación.
Para ampliar información sobre este tema, puede acudirse al libro "Patrones de Diseño" de E. Gamma et al., Ed. Addison-Wesley.
Para ampliar información sobre los patrones que se suelen usar en aplicaciones J2EE :

2. Estructura de los patrones

Los patrones usualmente se describen con la siguiente información:

- Descripción del problema: Que permitirá el patrón y ejemplos de situaciones del mundo real.
- Consideraciones: Que aspectos fueron considerados para que tomara forma esta solución.
- Solución general: Una descripción básica de la solución en si misma.
- Consecuencias: Cuales son los pros y contras de utilizar esta solución.
- Patrones relacionados: Otros patrones con uso similar que deben ser considerados como alternativa.

3. Tipos de Patrones

Existen diferentes tipos de patrones. Dependiendo del nivel conceptual de desarrollo donde se apliquen, se distinguen (de más abstractos a más concretos): patrones de análisis, patrones arquitectónicos, patrones de diseño y patrones de implementación o *idioms*.
Dependiendo del propósito funcional del patrón, se distinguen los siguientes tipos:

- Fundamental: construye bloques de otros patrones.
- Presentación: Estandariza la visualización de datos.
- De creación: Creación condicional de objetos.
- Integración: Comunicación con aplicaciones y sistemas y recursos externos.
- De particionamiento: Organización y separación de la lógica compleja, conceptos y actores en múltiples clases.
- Estructural: Separa presentación, estructuras de datos, lógica de negocio y procesamiento de eventos en bloques funcionales.
- De comportamiento: Coordina/Organiza el estado de los objetos.
- De concurrencia: Maneja el acceso concurrente de recursos.

NOTA:
Las aplicaciones Web pueden desarrollarse utilizando cualquier arquitectura posible.
En este caso analizaremos el desarrollo de aplicaciones mediante el patrón MVC.

Patrón MVC

El patrón fue descrito por primera vez en 1979[1] por Trygve Reenskaug, entonces trabajando en Smalltalk en laboratorios de investigación de Xerox.
La implementación original esta descrita a fondo en Programación de Aplicaciones en Smalltalk-80: Como utilizar Modelo Vista Controlador.
El Model-View-Controller (Modelo-Vista-Controlador) es designado habitualmente como MVC y posee diferentes clasificaciones:
Según Pattern Oriented Software Architecture, publicado por Buschmann en 1996, MVC se sitúa en la tercera de las cuatro categorías:

- Del barro a la estructura (From mud to Structure)
- Sistemas Distribuidos (Distributed Systems)
- Sistemas Interactivos (Interactive Systems)
- Sistemas Adaptables (Adaptable Systems)

Según Pattern of Enterprise Application Architecture, descrito recientemente por Fowler en 2003, el patrón se ubica en la tercera de las siete categorías:

- Patrones de lógica del dominio (Domain Logia Patterns)
- Patrones de Mapeo a Bases de Datos Relacionales (Mapping to Relational Database Patterns)
- Patrones de Presentación Web (Web Presentation Patterns)
- Patrones de Distribución (Distribution Patterns)
- Patrones de Concurrencia Offline (Offline Concurrency Patterns)
- Patrones de Estado de Sesión (Session State Patterns)
- Patrones Base (Base Patterns)

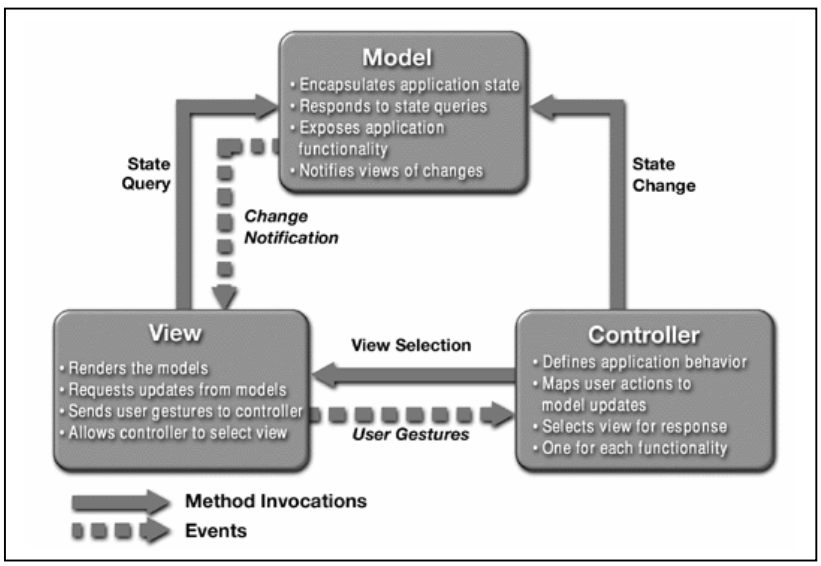
En general se acepta que el patrón Modelo-Vista-Controlador (MVC) es un ejemplo de patrón arquitectónico estructural que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.
La ventaja del patrón, por así decirlo, es una suerte de "separación en tres capas", donde el sistema está dividido en partes con tres responsabilidades completamente distintas, pero que interactúan constantemente entre ellas.
Estas capas o áreas responden básicamente a procesamiento, salida y entrada. Para esto, utiliza las siguientes abstracciones:

- Modelo (Model): Encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y/o comportamiento de entrada. Es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos; por ejemplo, no permitiendo comprar un número de unidades negativo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o importes en un carrito de la compra. Po lo tanto, el modelo es todo acceso a base de datos, y las funciones que incorporan la "lógica de negocio". O sea, las funciones más "pequeñas" con significado: ingresar una cantidad, obtener un listado de historias, etc.
- Vista (View): Muestra la información al usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador. Esta presenta el modelo en un formato adecuado para interactuar, usualmente es la interfaz de usuario. La vista, en una aplicación web, es el HTML (o XML, WAP, PDF, ...) y lo necesario para convertir datos en HTML. El acceso a bases de datos no es vista. La gestión de sesiones no es vista.
- Controlador (Controller): Reciben las entradas, usualmente como eventos que codifican los movimientos o pulsación de botones del ratón, pulsaciones de teclas, etc. Los eventos son traducidos a solicitudes de servicio ("service requests") para

el modelo o la vista, esto es se invocan a cambios en el modelo y probablemente en la vista. El controlador es lo que une la vista y el modelo. Por ejemplo, son las funciones que toman los valores de un formulario, consultan la base de datos (a través del modelo) y producen valores, que la vista tomará y convertirá en HTML.

La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones.
Si por ejemplo una misma aplicación debe ejecutarse tanto en un navegador estándar como un un navegador de un dispositivo móvil, solamente es necesario crear una vista nueva para cada dispositivo; manteniendo el controlador y el modelo original. El controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email, etc.). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación.
El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. Luego de mucho tiempo de probar y experimentar, se ha llegado a la conclusión que este tipo de "esquema central" son los más adecuados para el desarrollo de aplicaciones web, donde siempre se repite el siguiente comportamiento: 'Alguien, externo, solicita "algo" (aquí se hace responsable el "controlador"), recibiendo todos los pedidos y derivando a quién de los siguientes corresponda, la Vista o el Modelo'
Muchos sistemas informáticos utilizan un Sistema de Gestión de Base de Datos para gestionar los datos. En MVC corresponde al modelo.

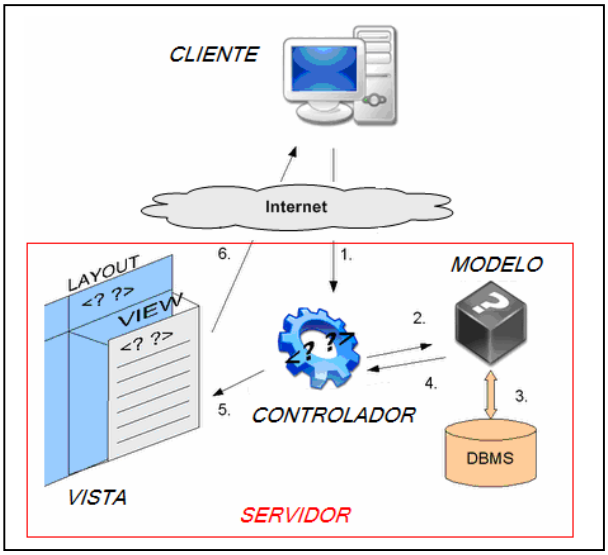
Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente:



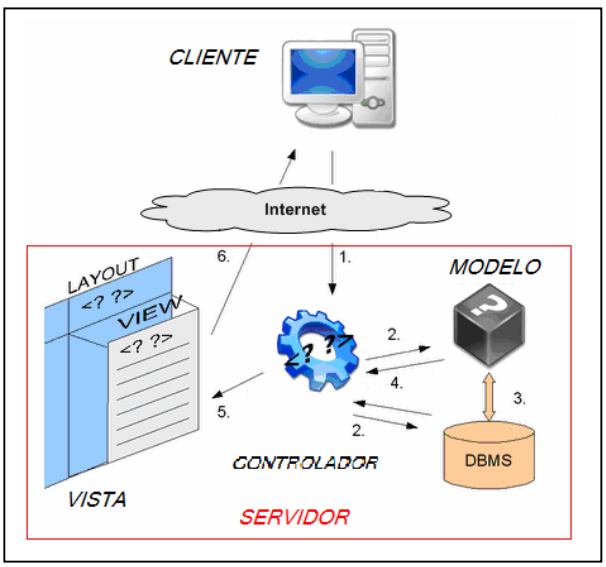
1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace)
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador puede ser utilizado para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.
5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

En un alto porcentaje de implementaciones concretas es posible encontrar MVC modificados, como se muestra en las figuras siguientes:

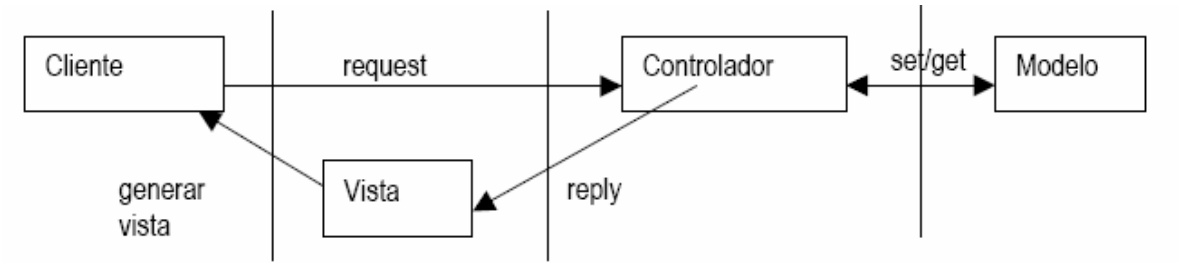


En este caso la capa de control recibe eventos de manera directa (peticiones) que deberán ser procesados (interaccion con la capa modelo) para generar o enviar a la capa de vista los datos o resultados que correspondan.

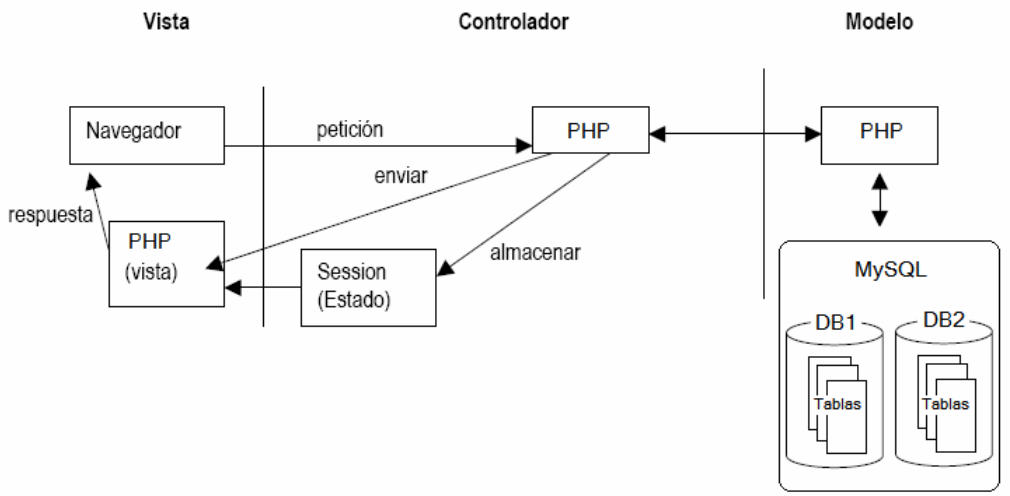


En este caso se observa que la capa de control también interactúa con los datos (pasos 2 y 4).
En general, se considera que esta última arquitectura no representa la mejor implementación o solución del patrón.

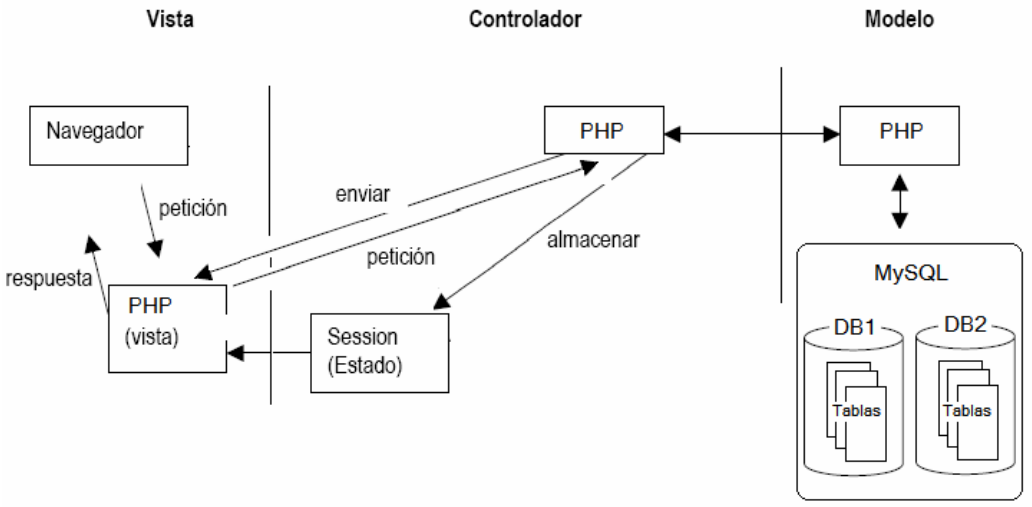
El modelo genérico desde una perspectiva de capas de desarrollo de software, se muestra a continuación:



Para el caso particular de nuestro trabajo, los esquemas posibles a usar son:
A) El modelo estándar en que el controlador es mediador entre modelo y vista.



B) El modelo modificado, donde los eventos en el browser realizan las invocaciones a las capas de vista, las que son redireccionadas a los controladores. Esto incorpora un salto más. Habrá que analizar la conveniencia de su utilización.



MVC en tres pasos, en PHP y sin objetos

1. M de Modelo

Las funciones "con significado" que mencioné antes deberían estar agrupadas lógicamente. O sea, no de una "forma lógica", sino que las que sean parecidas estén juntas.

El modelo incluye todas las funciones necesarias para acceder a bases de datos, recursos de la máquina, etc.

Estas funciones son de acceso a base de datos. Son funciones muy simples, que por si solas no tienen "significado" fuera de la base de datos:

```
function db_quitar_dinero ($cuenta, $cantidad)
function db_contar_historias ()
function db_buscar_productos_de_fabricante ($fabricante)
function db_borrar_usuario ($usuario)
```

Estas funciones tienen "significado": son operaciones "reales" sobre cosas "reales":

```
function obtener_historias ($num)
function transferir_dinero ($cantidad, $orig, $dest)
function añadir_carrito ($producto, $unidades, $usuario)
```

La principal diferencia es que las primeras son simples operaciones sobre la base de datos, y que las segundas pueden componerse de una o varias llamadas a funciones de acceso a base de datos y, posiblemente, cierta cantidad de "procesamiento".

Ejemplo:

```
function db_obtener_productos () {
   $prod = array();
   mysql_pconnect (DBHOST, DBUSER, DBPASS);
   mysql_select_db (DBNAME);
   $query = "SELECT id, nombre, cantidad, descripcion FROM productos";
   $result = mysql_query ($query);
   while ($row = mysql_fetch_array ($result)) {
            array_push ($prod, $row);
   }
   return $prod;
}

function obtener_productos ($letra) {
   if ($letra == "") {
            return db_obtener_productos ();
   }
   else {
            return db_obtener_productos_con_letra ($letra);
   }
}
```

Razones

Cada acceso a base de datos se pone en su función individual porque, de esta forma, si se cambia de gestor de bases de datos este cambio sólo afecta a estas funciones, no al resto de la aplicación.

Tener el modelo bien delimitado permite la existencia de varias aplicaciones que compartan el mismo modelo (por ejemplo, una aplicación "tienda" y una "contabilidad" que accedan a las bases de datos de inventario, ventas, ...)

2. C de Controlador

Como se dijo, el controlador no accede (directamente) a la base de datos, ni genera (directamente) HTML; se limita a obtener valores, procesarlos y obtener otros valores (además de la gestión de sesiones, cookies, logs y ese tipo de tonterías). Lo típico es recoger los valores de un formulario, procesarlos, trabajar un poco con la base de datos, procesar algo más, y almacenar el resultado en una o varias variables:

```
$letra = $POST["letra"];
if (($letra < "A") || ($letra > "Z")) {
        include "letranovalida.php";
}
else {
        $productos = obtener_productos ($letra);
        include "listadoproductos.php";
}
```

letranovalida.php y listadoproductos.php son parte de la vista. El segundo fichero hace uso de la variable $productos para mostrar el listado. Como se puede ver, el controlador no se mete con HTML; se limita a obtener el listado de la base de datos y pasárselo, tal cual, a la vista.

Razones
De este modo, el código que "hace algo" está perfectamente separado del código dedicado a crear HTML, lo que ayuda a evitar el *spaghetti*.

3. V de Vista
La vista es la parte de la aplicación dedicada a generar HTML. Puede ser simple HTML estático:

```
<h1>Listado de productos</h1>
```

O HTML dinámico generado en respuesta a una consulta, por ejemplo:

```
<table>
        <?php
                foreach ($productos as $p) {
        ?>
        <tr><td>
        <a href="verproducto.php?id=<?=$p["id"]?>"><?=$p["nombre"]?></a>
        </td>
        <td> <?=$p["cantidad"]?> </td>
        </tr>
        <?php
                }
        ?>
</table>
```

La vista no debería llamar a ninguna función; debería limitarse a construcciones de control de flujo simple: if, for/foreach, while, ...
Todas las variables empleadas por la vista deberían proceder del controlador; utilizar directamente variables procedentes del usuario es una práctica muy peligrosa.

Razones
Tener la vista separada del controlador permite cambiar la aplicación para que genere, en lugar de HTML, algo distinto (por ejemplo, WML), sin tener que tocar más que una parte completamente delimitada del código.

Transformación de una 'aplicación plana' en una 'aplicación MVC' con PHP

Para poder entender las ventajas de utilizar el patrón MVC, se va a transformar una aplicación simple (plana) realizada con PHP en una aplicación que sigue la arquitectura MVC.
Para ilustrar esta explicación se considerará la presentación de una lista con las últimas entradas o artículos de un blog.

Programación simple en PHP
Utilizando solamente PHP normal y corriente, el script necesario para mostrar los artículos almacenados en una base de datos se muestra en el siguiente listado 1:

```
<?php

// Conectar con la base de datos y seleccionarla
$conexion = mysql_connect('localhost', 'miusuario', 'micontrasena');
mysql_select_db('blog_db', $conexion);

// Ejecutar la consulta SQL
$resultado = mysql_query('SELECT fecha, titulo FROM articulo', $conexion);

?>

<html>
  <head>
    <title>Listado de Artículos</title>
  </head>
  <body>
   <h1>Listado de Artículos</h1>
   <table>
     <tr><th>Fecha</th><th>Titulo</th></tr>
<?php
// Mostrar los resultados con HTML
while ($fila = mysql_fetch_array($resultado, MYSQL_ASSOC))
{
echo "\t<tr>\n";
printf("\t\t<td> %s </td>\n", $fila['fecha']);
printf("\t\t<td> %s </td>\n", $fila['titulo']);
echo "\t</tr>\n";
}
?>
    </table>
  </body>
</html>

<?php

// Cerrar la conexion
mysql_close($conexion);

?>
```

El script anterior es fácil de escribir y rápido de ejecutar, pero muy difícil de mantener y actualizar. Los principales problemas del código anterior son:

- La incorporación del control de errores en el código complica la comprensión y secuencialidad general (es el caso de un fallo de conexión con la base de datos).
- El código HTML y el código PHP están mezclados en el mismo archivo e incluso en algunas partes están entrelazados.
- El código solo funciona si la base de datos es MySQL.

Las capas de la arquitectura MVC

Separando la presentación

Las llamadas a echo y printf del listado 1 dificultan la lectura del código. De hecho, modificar el código HTML del script anterior para mejorar la presentación es un follón debido a cómo está programado. Así que el código va a ser dividido en dos partes.
En primer lugar, el código PHP puro con toda la lógica de negocio se incluye en el script del controlador, como se muestra en el listado 2: index.php

```
<?php
// Conectar con la base de datos y seleccionarla
$conexion = mysql_connect('localhost', 'miusuario', 'micontrasena');
mysql_select_db('blog_db', $conexion);

// Ejecutar la consulta SQL
$resultado = mysql_query('SELECT fecha, titulo FROM articulo', $conexion);

// Crear el array de elementos para la capa de la vista
$articulos = array();
while ($fila = mysql_fetch_array($resultado, MYSQL_ASSOC))
{
  $articulos[] = $fila;
}

// Cerrar la conexión
mysql_close($conexion);

// Incluir la lógica de la vista
require('vista.php');
?>
```

El código HTML, que contiene cierto código PHP a modo de plantilla, se almacena en el script de la vista, como se muestra en el listado 3: vista.php

```
<html>
  <head>
    <title>Listado de Artículos</title>
  </head>
  <body>
    <h1>Listado de Artículos</h1>
    <table>
      <tr><th>Fecha</th><th>Título</th></tr>
    <?php foreach ($articulos as $articulo): ?>
      <tr>
        <td><?php echo $articulo['fecha'] ?></td>
        <td><?php echo $articulo['titulo'] ?></td>
      </tr>
    <?php endforeach; ?>
    </table>
  </body>
</html>
```

Una buena regla general para determinar si la parte de la vista está suficientemente *limpia* de código es que debería contener una cantidad mínima de código PHP, la suficiente como para que un diseñador HTML sin conocimientos de PHP pueda entenderla. Las instrucciones más comunes en la parte de la vista suelen ser echo, if/else, foreach/endforeach y poco más. Además, no se deben incluir instrucciones PHP que generen etiquetas HTML.
Toda la lógica se ha centralizado en el script del controlador, que solamente contiene código PHP y ningún tipo de HTML. De hecho, y como puedes imaginar, el mismo

controlador se puede reutilizar para otros tipos de presentaciones completamente diferentes, como por ejemplo un archivo PDF o una estructura de tipo XML.

Separando la manipulación de los datos
La mayor parte del script del controlador se encarga de la manipulación de los datos. Pero, ¿qué ocurre si se necesita la lista de entradas del blog para otro controlador, por ejemplo uno que se dedica a generar el canal RSS de las entradas del blog? ¿Y si se quieren centralizar todas las consultas a la base de datos en un único sitio para evitar duplicidades? ¿Qué ocurre si cambia el modelo de datos y la tabla articulo pasa a llamarse articulo_blog? ¿Y si se quiere cambiar a PostgreSQL en vez de MySQL?
Para poder hacer todo esto, es imprescindible eliminar del controlador todo el código que se encarga de la manipulación de los datos y ponerlo en otro script, llamado *el modelo*, tal y como se muestra en el listado 4: modelo.php

```
<?php
function getTodosLosArticulos()
{
  // Conectar con la base de datos y seleccionarla
  $conexion = mysql_connect('localhost', 'miusuario', 'micontrasena');
  mysql_select_db('blog_db', $conexion);

  // Ejecutar la consulta SQL
  $resultado = mysql_query('SELECT fecha, titulo FROM articulo', $conexion);

  // Crear el array de elementos para la capa de la vista
  $articulos = array();
  while ($fila = mysql_fetch_array($resultado, MYSQL_ASSOC))
  {
    $articulos[] = $fila;
  }

  // Cerrar la conexión
  mysql_close($conexion);

  return $articulos;
}

?>
```

El controlador modificado se puede ver en el listado 5: index.php

```
<?php
// Incluir la lógica del modelo
require_once('modelo.php');

// Obtener la lista de artículos
$articulos = getTodosLosArticulos();

// Incluir la lógica de la vista
require('vista.php');
?>
```

Ahora el controlador es mucho más fácil de leer. Su única tarea es la de obtener los datos del modelo y pasárselos a la vista. En las aplicaciones más complejas, el controlador se encarga además de procesar las peticiones, las sesiones de los usuarios, la autenticación, etc. El uso de nombres apropiados para las funciones del modelo hacen que sea innecesario añadir comentarios al código del controlador.

El script del modelo solamente se encarga del acceso a los datos y puede ser reorganizado a tal efecto. Todos los parámetros que no dependen de la capa de datos (como por ejemplo los parámetros de la petición del usuario) se deben obtener a través del controlador y por tanto, no se puede acceder a ellos directamente desde el modelo. Las funciones del modelo se pueden reutilizar fácilmente en otros controladores.

Separación en capas más allá del MVC

El principio más importante de la arquitectura MVC es la separación del código del programa en tres capas, dependiendo de su naturaleza. La lógica relacionada con los datos se incluye en el modelo, el código de la presentación en la vista y la lógica de la aplicación en el controlador.

La programación se puede simplificar si se utilizan otros patrones de diseño. De esta forma, las capas del modelo, la vista y el controlador se pueden subidividir en más capas.

Abstracción de la base de datos

La capa del modelo se puede dividir en la capa de acceso a los datos y en la capa de abstracción de la base de datos. De esta forma, las funciones que acceden a los datos no utilizan sentencias ni consultas que dependen de una base de datos, sino que utilizan otras funciones para realizar las consultas. Así, si se cambia de sistema gestor de bases de datos, solamente es necesario actualizar la capa de abstracción de la base de datos. El listado 6 muestra una capa de acceso a datos específica para MySQL y el listado 7 muestra una capa sencilla de abstracción de la base de datos.

Listado 6 - Parte del modelo de abstracción de la base de datos:

```
<?php
function crear_conexion($servidor, $usuario, $contrasena)
{
  return mysql_connect($servidor, $usuario, $contrasena);
}

function cerrar_conexion($conexion)
{
  mysql_close($conexion);
}

function consulta_base_de_datos($consulta, $base_datos, $conexion)
{
  mysql_select_db($base_datos, $conexion);

  return mysql_query($consulta, $conexion);
}

function obtener_resultados($resultado)
{
  return mysql_fetch_array($resultado, MYSQL_ASSOC);
}
```

Listado 7 - Parte del modelo correspondiente al acceso a datos:

```
function getTodosLosArticulos()
{
  // Conectar con la base de datos
  $conexion = crear_conexion('localhost', 'miusuario', 'micontrasena');
```

```
  // Ejecutar la consulta SQL
  $resultado = consulta_base_de_datos('SELECT fecha, titulo FROM articulo', 'blog_db', $conexion);

  // Crear el array de elementos para la capa de la vista
  $articulos = array();
  while ($fila = obtener_resultados($resultado))
  {
    $articulos[] = $fila;
  }

  // Cerrar la conexión
  cerrar_conexion($conexion);

  return $articulos;
}
?>
```

Como se puede comprobar, la capa de acceso a datos no contiene funciones dependientes de ningún sistema gestor de bases de datos, por lo que es independiente de la base de datos utilizada. Además, las funciones creadas en la capa de abstracción de la base de datos se pueden reutilizar en otras funciones del modelo que necesiten acceder a la base de datos.

Nota:
Los ejemplos de los listados 6 y 7 no son completos, y todavía hace falta añadir algo de código para tener una completa abstracción de la base de datos (abstraer el código SQL mediante un constructor de consultas independiente de la base de datos, añadir todas las funciones a una clase, etc.)

Los elementos de la vista
La capa de la vista también puede aprovechar la separación de código. Las páginas web suelen contener elementos que se muestran de forma idéntica a lo largo de toda la aplicación: cabeceras de la página, el *layout* genérico, el pie de página y la navegación global. Normalmente sólo cambia el interior de la página. Por este motivo, la vista se separa en un layout y en una plantilla. Normalmente, el layout es global en toda la aplicación o al menos en un grupo de páginas. La plantilla sólo se encarga de visualizar las variables definidas en el controlador. Para que estos componentes interaccionen entre sí correctamente, es necesario añadir cierto código.
Siguiendo estos principios, la parte de la vista del listado 3 se puede separar en tres partes, como se muestra en los listados 8, 9 y 10.

Listado 8 - La parte de la plantilla de la vista, en miplantilla.php

```
<h1>Listado de Artículos</h1>
<table>
<tr><th>Fecha</th><th>Título</th></tr>
<?php foreach ($articulos as $articulo): ?>
  <tr>
    <td><?php echo $articulo['fecha'] ?></td>
    <td><?php echo $articulo['titulo'] ?></td>
  </tr>
<?php endforeach; ?>
</table>
```

Listado 9 - La parte de la lógica de la vista

```
<?php
$titulo = 'Listado de Artículos';
$contenido = include('miplantilla.php');
?>
```

Listado 10 - La parte del layout de la vista

```
<html>
  <head>
    <title><?php echo $titulo ?></title>
  </head>
  <body>
    <?php echo $contenido ?>
  </body>
</html>
```

Acciones y controlador frontal

En el ejemplo anterior, el controlador no se encargaba de realizar muchas tareas, pero en las aplicaciones web reales el controlador suele tener mucho trabajo. Una parte importante de su trabajo es común a todos los controladores de la aplicación. Entre las tareas comunes se encuentran el manejo de las peticiones del usuario, el manejo de la seguridad, cargar la configuración de la aplicación y otras tareas similares. Por este motivo, el controlador normalmente se divide en un controlador frontal, que es único para cada aplicación, y las acciones, que incluyen el código específico del controlador de cada página.

Una de las principales ventajas de utilizar un controlador frontal es que ofrece un punto de entrada único para toda la aplicación. Así, en caso de que sea necesario impedir el acceso a la aplicación, solamente es necesario editar el script correspondiente al controlador frontal. Si la aplicación no dispone de controlador frontal, se debería modificar cada uno de los controladores.

Orientación a objetos

Los ejemplos anteriores utilizan la programación procedimental. Las posibilidades que ofrecen los lenguajes de programación modernos para trabajar con objetos permiten simplificar la programación, ya que los objetos pueden encapsular la lógica, pueden heredar métodos y atributos entre diferentes objetos y proporcionan una serie de convenciones claras sobre la forma de nombrar a los objetos.

La implementación de una arquitectura MVC en un lenguaje de programación que no está orientado a objetos puede encontrarse con problemas de *namespaces* y código duplicado, dificultando la lectura del código de la aplicación.

La orientación a objetos permite a los desarrolladores trabajar con objetos de la vista, objetos del controlador y clases del modelo, transformando las funciones de los ejemplos anteriores en métodos. Se trata de un requisito obligatorio para las arquitecturas de tipo MVC.

Referencias

http://java.sun.com/products/jfc/tsc/articles/architecture/
http://www.indiawebdevelopers.com/technology/java/mvcarchitecture.asp
http://www.ibm.com/developerworks/library/ws-mvc/
http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html
http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/app-arch/app-arch2.html
http://www.webtaller.com/construccion/lenguajes/php/lecciones
http://developer.java.sun.com/developer/technicalArticles/J2EE/patterns/
http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html

y oops: http://mundogris.wordpress.com/2008/02/17/mvc-no-es-una-buena-arquitectura-para-la-web/